EECS 3201 - Lab #4 Report
Shot Clock
Group 04 Section B:
Ajay Deonandan #217987215
Isaiah Gocool #218918052

Overview

Objective

In lab 4, we were tasked with creating a shot clock used in basketball, using sequential logic design. The LED hex displays on the DE-10 Lite board would be used to act as the clock. The clock could start at either 24 or 30 seconds depending on a switch that is pressed, and could be reset or paused using two different buttons. When the timer hits zero seconds, the decimal would flash every 0.5 seconds.

Acronyms

The 10 DE10-Lite toggle switches are referred to as SW[N] where N is the number of the switch being referred to. The 7-Segment displays are referred to as HEXN where N is the display number referred to.

Design Approach

Inputs and Outputs

On the DE10 Lite board, the HEX0 and HEX1 displays were used to display the countdown timer. HEX1 would display the first digit, while HEX0 would display the second. Button 1 was used to pause the timer, and button 2 was used to reset the timer. Switch 0 was also utilized to change the timer from 24 seconds to 30 seconds and vice-versa.

Pause Button

We used two modules for this lab; ClockDivider and lab4. ClockDivider was used to calculate the amount of time that passes, which in this case would be either 1 second or 0.5 seconds. Lab4 was used to change the values on the HEX displays, so it would display the current time in the countdown. To check for when the button is pressed, we used an always@ block in ClockDivider, that would check for the condition “negedge ~button[1]”. “Negedge” must be used because it detects the negative edge of the button, which is when the button press is released. “Button[1]” needs to be negated because the button outputs a value of 0 when it is pressed, and 1 when it is released. Inside this always@ block, we set “pause = ~pause”, to signify that the timer needs to be stopped.

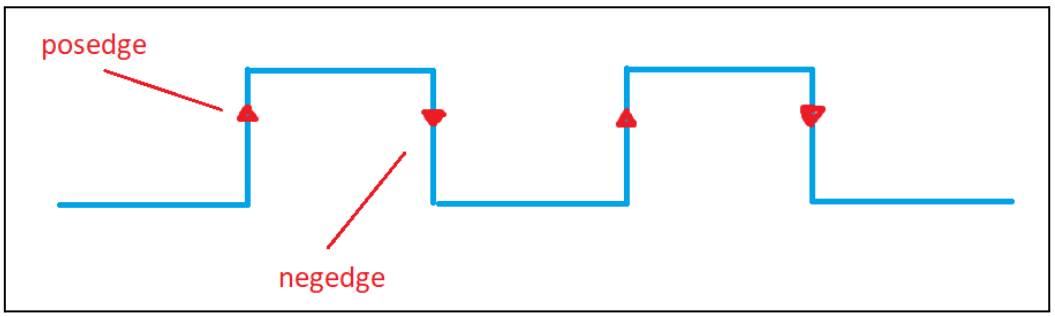

Figure 1: Diagram of the clock signal, and where the negative (negedge) and positive (posedge) edges are.

Countdown Timer

Our ClockDivider module utilized the DE-10’s clock 1 which provides a 50MHz clock input. The posedge of this clock was used in order to count down each second, count every 0.5 seconds as well as to handle the reset input. Inside our clock’s always @ block, we had a counter that consistently was incremented. Once this counter reached a decimal value of 50000000, this represented that 50 million cycles have gone by. Since our clock has a frequency of 50MHz, if we divide our frequency by the number of cycles that have gone by we can get the amount of time that has elapsed. In this case

everytime our counter gets incremented 50 million times (counter = 50 million), one second goes by. We then had an if statement checking to see if our counter has been incremented to 50 million and once it has (symbolizing that one second has gone by) we subtracted 1 from a decrementer. This would represent the time of the shot clock which is decreased by 1 each second starting from the value of 24. Afterwards, we reset this counter to 0 for it to count up to 50 million again to determine when the next second will occur. This if statement also only runs when our pause reg is set to 0. This means that if our pause button is pressed to complement the pause reg, our decrementer shouldn't be subtracted every second. We only are able to subtract from it if our pause variable is equal to 1.

Flashing Dot

We had a similar set up in this always @ block using a separate counter, this time having an if statement checking to see if the counter has been incremented to 25 million. Since we have a clock of 50MHz (50,000,000) and we are checking for 25 million cycles, everytime this counter is incremented to 25 million we know that 0.5 seconds goes by. If we determine that this counter reached 25 million we complement an output 'flasher' reg which is used in our main module to blink the dot on the hex display every 0.5 seconds when our subtracted equals zero. We also then reset this counter to 0 for it to start counting back up to 25 million again.

Reset Button

Inside of this always @ block we also had an if statement comparing the state of button 0 and our flip switch. If our flip switch was flipped downwards and our button had been pressed, we would set our decrementer to 24. If the switch was flipped upwards and our button was pressed we would reset our decrementer to 30. This allowed us to have a way to reset our timer when a button is pressed and also have a way to switch between starting at 30 or 24 seconds.

Initialization of Countdown Timer

There is also logic to set the initial value of our decrementer to 24 or 30 based on the initial state of the flip switch. We have an if-else statement to check that if our decrementer's value is equal to 24 whether our switch is in the upwards position (representing to start at 30), which tells us to set the decrementer to 30. Then, if our switch is in the downwards position and our decrementer is 30, set our decrementer to 24. We then also set our initial flag to 0 as both of these if statements would also only work if our initial flag was equal to 1. This allowed this initial decrementer set up to only work once (when the code is blasted onto the FPGA).

Displaying Timer on Segment Displays

Our main method runs the ClockDivider module and also has an always @ block that uses our decrementer as an event. This always @ block contains a case that is used to display our decrementers value onto HEX0 and HEX1. Our case statement would take in our decrementer as a condition (a binary value) and compare it against 0 to 30 (in binary). Depending on what number our decrementer is, we set two reg's - f and g - equal to a corresponding binary number that is used to display a certain number onto our 7 segment displays. The following table tells us which 7 segment display LEDs must be turned on and off to display a corresponding number.

HEX0[0]	HEX0[1]	HEX0[2]	HEX0[3]	HEX0[4]	HEX0[5]	HEX0[6]	HEX0[7]	
0	0	0	0	0	0	1	1	**0**
1	0	0	1	1	1	1	1	**1**
0	0	1	0	0	1	0	1	**2**
0	0	0	0	1	1	0	1	**3**
1	0	0	1	1	0	0	1	**4**
0	1	0	0	1	0	0	1	**5**
0	1	0	0	0	0	0	1	**6**
0	0	0	1	1	1	1	1	**7**
0	0	0	0	0	0	0	1	**8**
0	0	0	0	1	0	0	1	**9**

Figure 2: Truth table for displaying each integer on the seven-segment display.

Reg f would be set to the binary string that creates the tens place value of our decrementer and reg g would be set to the binary string that creates the ones place value of our decrementer. For example, if our decrementer was equal to 15, f would be set to the binary number that creates 1 on our segment display and g would be set to the binary number that creates 5 on our segment display. Once these reg's are set, we then have several assign statements to assign each led for HEX0 and HEX1's LEDs to the values of f and g where g is displayed on HEX0 (the display to the right) and f is displayed on HEX1 (the display left of HEX0). This creates the display of a 2 digit decimal number.

Inside of our main module always @ block, we also have an if statement checking to see if our decrementer is equal to 0. When our decrementer is 0, we take our flash reg that is being complemented every 0.5 seconds in our clock divider and assign it to the dot on our segment display which causes the dot LED to flash every 0.5 seconds. If our decrementer is reset and doesn't equal 0 anymore, we set our dot LED back to 1 to turn it off.

Test Strategy

Final Testing Strategy

To start, we implemented the ClockDivider module included in the lab manual, and then tried testing it to see what results we get and how we can manipulate it. This allowed us to get a greater understanding of how the module worked, and what we needed to change in order to make it act like the clock used in a basketball game. Once we deemed our code finished we came up with several test cases to cover every feature required by the lab manual. We first started with checking to see if our counter would count down to 0 from 24 every second once the board was blasted.

After determining that this worked we moved onto finding out if the dot on HEX0 flashed every 0.5 seconds whenever our display said 0. To determine that the dot was flashing every 0.5 seconds and the display counted down by 1 every second, we opened up a stopwatch on a mobile device and compared the timings to our countdown. This allowed us to verify that there was an interval exactly one second between each decrement for our countdown as well as a 0.5 second interval between when our dot on HEX0 flashed on and off.

Having verified that the timings were correct, we moved onto testing to see if our reset button worked. We tested whether our reset button resetted our countdown at random times between 24 and 1 and we also made sure we could reset our countdown at 0 after some time had passed.

Once we confirmed the reset button worked, we then tested to see if we could switch between 30 seconds and 24 seconds using switch 0 on the DE-10 Lite board. We made sure that once the switch was in the upwards position and the reset button was pressed, our countdown would start from 30 and if we flipped our switch back and pressed the reset button it would reset our countdown back to 24. We also made sure our countdown counted all the way down from 30 to 0, the dot on HEX0 flashed every 0.5 seconds when counting down from 30 and if we could reset the timer back to 30 when counting down from 30.

We then tested our pause button to make sure that our timer would pause at random times from 24 - 1 as well as times 30 - 1. When our countdown is 0, we also made sure to see whether our pause button would freeze the dot flashing on HEX0 as well as if we were able to reset our timer while paused.

Afterwards, we tested to see if our countdown would start from 30 or 24 depending on the state of switch 0 when our code is initially blasted to the fpga board.

Issues Encountered

One problem we encountered after we finished writing our code, was that we sometimes had to press button[1] multiple times in order to pause it, when one press should have been sufficient. The number of times we had to press the button was inconsistent as well, ranging from 2 to 5 presses. Our first thought was that the button on the board could be broken, so we decided to test our hypothesis by creating a new project where the LED on the board would be turned on with the push of the button. When we tested this new program, we found that the button worked perfectly fine, so we knew that there was a problem with the logic in our code. Since it was only the pause button that wasn't working, we knew that it must be a problem with how we wrote our always@ block to check for when the button was pressed. Upon further research, we found out that the problem was that we were looking for the posedge of the clock, not the negedge.